LumiMAS: A Comprehensive Framework for Real-Time Monitoring and Enhanced Observability in Multi-Agent Systems

AAAI Track

Ron Solomon
Ben-Gurion University of the Negev
Beer Sheva, Israel
ronso@post.bgu.ac.il

Yarin Yerushalmi Levi
Ben-Gurion University of the Negev
Beer Sheva, Israel
yarinye@post.bgu.ac.il

Lior Vaknin
Ben-Gurion University of the Negev
Beer Sheva, Israel
liorva@post.bgu.ac.il

Eran Aizikovich
Ben-Gurion University of the Negev
Beer Sheva, Israel
eranaizi@post.bgu.ac.il

Amit Baras
Ben-Gurion University of the Negev
Beer Sheva, Israel
barasa@post.bgu.ac.il

Etai Ohana
Ben-Gurion University of the Negev
Beer Sheva, Israel
etaiohana@bgu.ac.il

Amit Giloni
Fujitsu Research of Europe
Modiin, Israel
amit.giloni@fujitsu.com

Shamik Bose
Fujitsu Research of Europe
Leeds, UK
shamik.bose@fujitsu.com

Chiara Picardi
Fujitsu Research of Europe
York, UK
chiara.picardi@fujitsu.com

Yuval Elovici
Ben-Gurion University of the Negev
Beer Sheva, Israel
elovici@bgu.ac.il

Asaf Shabtai
Ben-Gurion University of the Negev
Beer Sheva, Israel
shabtaia@bgu.ac.il

ABSTRACT

The incorporation of large language models in multi-agent systems (MASs) has the potential to significantly improve our ability to autonomously solve complex problems. However, such systems introduce unique challenges in monitoring, interpreting, and detecting system failures. Most existing MAS observability frameworks focus on analyzing each individual agent separately, overlooking failures associated with the entire MAS. To bridge this gap, we propose LumiMAS, a novel MAS observability framework that incorporates advanced analytics and monitoring techniques. The proposed framework consists of three key components: a monitoring and logging layer, anomaly detection layer, and anomaly explanation layer. LumiMAS's first layer monitors MAS executions, creating detailed logs of the agents' activity. These logs serve as input to the anomaly detection layer, which detects anomalies across the MAS workflow in real time. Then, the anomaly explanation layer performs classification and root cause analysis (RCA) of the detected anomalies. LumiMAS was evaluated on seven different MAS applications, implemented using two popular MAS platforms, and a diverse set of possible failures. The applications include two novel failure-tailored applications that illustrate the effects of a hallucination or bias on the MAS. The evaluation results demonstrate LumiMAS's effectiveness in failure detection, classification, and RCA.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/PNWL9707

KEYWORDS

AI Agents; AI Security; Anomaly Detection; Observability

ACM Reference Format:
Ron Solomon, Yarin Yerushalmi Levi, Lior Vaknin, Eran Aizikovich, Amit Baras, Etai Ohana, Amit Giloni, Shamik Bose, Chiara Picardi, Yuval Elovici, and Asaf Shabtai. 2026. LumiMAS: A Comprehensive Framework for Real-Time Monitoring and Enhanced Observability in Multi-Agent Systems: AAAI Track. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/PNWL9707

1 INTRODUCTION

The incorporation of large language models (LLMs) in multi-agent systems (MASs) is rapidly reshaping intelligent systems by enabling dynamic context-aware interactions across real-world applications [13]. For instance, MetaGPT [16] demonstrates how LLM-based MASs can coordinate effectively to solve complex coding tasks. Another example is PMC [36], which assigns constraint-specific planning agents that solve real-world problems such as travel planning. However, the incorporation of LLMs makes MASs susceptible to the vulnerabilities inherent in these models, such as (1) hallucinations [19], (2) biases [24], and (3) adversarial manipulations [11] that threaten the trustworthiness of AI systems. Since errors can propagate through inter-agent interactions, such issues are often amplified in MASs [2]. Furthermore, the autonomous and dynamic nature of MASs exposes them to risks highlighted by the OWASP Top 10 for LLM Applications Project [26], including prompt injections, memory poisoning, and cascading hallucinations. Notably, the 2025 OWASP list underscores the importance of robust monitoring solutions and Srikumar et al. [32] highlight the

need for real-time failure detection solutions in managing agentic AI risks. Nowadays, most existing monitoring and failure detection approaches analyze each agent separately, utilize LLM that induce latency and computational cost to the system, or focus on a limited set of possible failures, which ultimately harm the relevance of the solution, limiting their effectiveness for real-time, multi-user systems, and are not aligned with the users' needs. To address this gap, we present *LumiMAS*, a novel agnostic and efficient framework that incorporates advanced analytics and monitoring techniques, comprehensively capturing both system-level features and the semantic nuances of inter-agent interactions. LumiMAS utilize existing techniques with engineered semantic and operational features to effectively address the open problem of real-time MAS failure detection. It is designed to enhance observability, improve monitoring capabilities, enable real-time failure detection with minimal resource consumption, and supports the identification of a diverse range of system failures, as well as to improve the model's alignment with the user's needs. LumiMAS is composed of three key components: (1) a monitoring layer that performs system-wide logging across diverse MAS frameworks (extracting key operational and communication features); (2) an anomaly detection layer which identifies deviations from normal behavior in real-time; and (3) an anomaly explanation layer that consists of anomaly classification and root cause analysis (RCA) LLM-based agents (LMAs) whose aim is to identify the anomaly's type and source.

2 RELATED WORK

Despite the rapid adoption of LLM-based MASs, limited research has been conducted to assess their risks, mitigate failures, and monitor their workflow. However, the increased use of LMAs exposes the system they operate within to new threats, highlighting the need for robust tools to monitor and validate their actions effectively.

2.1 MAS Risk Assessment

To enhance MAS safety evaluation, Ruan et al. [31] used LLMs to emulate sandbox environments, enabling pre-deployment testing of agents when interacting with new tools. The work was extended by Yuan et al. [35], who added ambiguous instructions to evaluate whether agents can distinguish safe from unsafe actions. Zhang et al. [37] introduced the agent security bench, which evaluates attacks and defenses on LMAs operating in a MAS. While these studies concentrated on assessing the safety of MASs, our approach monitors deployed MASs to improve system observability, capturing a broad range of execution features, and facilitating real-time failure detection.

2.2 Failure Mitigation Strategies

Using rule-based mitigation strategies for unsafe actions, AgentMonitor [23], TrustAgent [17], AgenTRIM [1], and the visibility framework of Chan et al. [3], continuously inspect the LMA's chain-of-thought and planned tool calls in real time, to score or block unsafe steps and enforce domain-specific safety policies. Leveraging the reasoning capabilities of LLMs, Fang et al. [10] introduced an approach to infer the agent's intent from its action sequence, which is then verified to align with the user's instructions. Recently Peigné et al. [28] highlight how malicious prompts can propagate across agents in MAS, and introduce individual-agent mitigation strategies demonstrating a trade-off between robustness and cooperation. However, unlike these approaches, which primarily focus on *individual agents* operating in a MAS, LumiMAS addresses critical inter-agent dynamics and system-level failure detection.

2.3 MAS Monitoring

AgentOps [8] lays the theoretical groundwork for observing LMAs by tracing key agent artifacts and their associated data throughout the agent's life cycle. Chan et al. [4] proposed a solution for monitoring individual agent inputs and outputs and applying real-time corrections. In addition, a detailed list of commercial observability tools (e.g., LangSmith, LangFuse) can be found in the supplementary material [1].

Our framework extends MAS observability beyond traditional system- and agent-level associated data by performing internal communications analysis, which allows our solution to effectively understand and identify issues that other approaches may struggle to address.

3 THREAT MODEL

LLM-based MASs, which incorporate multiple components, are vulnerable to various failures that either originate from inherent limitations of the LLM, such as hallucinations [19] and biases [24, 34], or result from adversarial interference [7, 26].

Adversarial goal. As described by Zhang et al. [37], the primary adversarial goal is to manipulate AI agents into performing actions that deviate from their intended behavior, potentially resulting in unsafe, unethical, or harmful outcomes. Adversaries may attempt to cause system failures, degrade overall system performance, extract sensitive information, or perform resource exhaustion.

Threat actors, their capabilities and attack vectors. Similar to Deng et al. [7] and Zhang et al. [37], we consider a diverse set of threat actors that reflect realistic risks in MAS. While all threat actors are assumed to have prior knowledge, which could have been obtained in a prior reconnaissance stage, they typically lack access to the internals of the underlying LLM (i.e., the LLM is treated as a black-box). The adversarial attack can be performed by malicious users who may utilize different attack vectors, including direct prompt injection (DPI) and indirect prompt injection (IPI). In a DPI attack, the threat actor manipulates the input prompt to inject malicious instructions, which can lead to resource exhaustion, propagation of misinformation, or insertion of backdoor commands to compromise the system's integrity. IPI attacks exploit the agent's reliance on external tools (e.g., web search), allowing adversaries to plant malicious instructions within these sources and cause unintended actions. The attack may also be executed by external data providers or privileged users who control data sources (e.g., retrieval-augmented generation (RAG) database). In that case, an attacker may perform a memory poisoning (MP) attack, in which the agent's memory source is contaminated, causing the agent to behave according to the attacker's intent.

[1] Supplementary material is available at https://arxiv.org/abs/2508.12412

3.1 Threat Taxonomy

According to Cemri et al. [2], who introduced the MAST taxonomy, multi-agent failures can be classified into three categories: Specification Issues, Inter-Agent Misalignment, and Verification or Termination Deficiencies. When viewed through this lens, each failure type aligns with specific categories and failure modes defined in the MAST taxonomy. Our examined failures are mostly aligned with the first two categories, with limited focus on the third (Task Verification). Under specification issues, the following modes are most relevant: FM-1.1 Disobey Task Specification – failure to adhere to task constraints or requirements (DPI exhaustion, IPI); FM-1.2 Disobey Role Specification – deviation from assigned responsibilities or confusion between roles (DPI backdoor); FM-1.3 Step Repetition – redundant task execution or looping (DPI exhaustion, IPI); and FM-1.4 Loss of Conversation History – truncation or overwriting of prior context (DPI misinformation, MP, hallucination). Under Inter-Agent Misalignment, the relevant modes include FM-2.1 Conversation Reset – dialogue restart with context loss (hallucination, IPI); FM-2.3 Task Derailment – deviation from the intended goal (DPI, IPI, MP, hallucination); FM-2.4 Information Withholding – failure to share critical data across agents (IPI, DPI); FM-2.5 Ignored Other Agent's Input – disregard of peer contributions (IPI, DPI, MP); and FM-2.6 Reasoning–Action Mismatch – inconsistency between reasoning and behavior (MP, hallucination). In our work, we primarily target failures in specification integrity and misalignment, while explicit verification-oriented mechanisms remain a promising area for future extension.

4 METHODOLOGY

To address the challenges in monitoring and interpreting MAS behavior and addressing the open problem of real-time failure detection, we propose a comprehensive, multi-layered framework to enhance system observability. This methodology comprises three key components: (1) a monitoring and logging layer, (2) an anomaly detection layer, and (3) an anomaly explanation layer consisting of anomaly classification and RCA techniques.

In LumiMAS's first phase, execution data is systematically collected, aiming to capture diverse low-level operational features and high-dimensional semantic information reflecting agent activities and reasoning. The collected data undergoes preprocessing, transforming it into structured formats suitable for quantitative analysis and enabling an understanding of complex system dynamics. LumiMAS's analytical techniques employed in the anomaly detection and explanation layers utilize this processed data.

Figure 1 presents a high-level view of our proposed observability framework, LumiMAS. Ultimately, this architecture aims to provide deep visibility and interpretable insights necessary to maintain system robustness and security.

4.1 System Logging Layer

A fundamental component of our observability framework is a comprehensive system logging layer. LumiMAS logs several unique predefined events that abstract away implementation-specific behaviors, ensuring compatibility with different MAS platforms. The content of each log encodes system behavior into a chronological sequence of structured events $\{e_1, e_2, \ldots, e_n\}$ where each e_i is an event

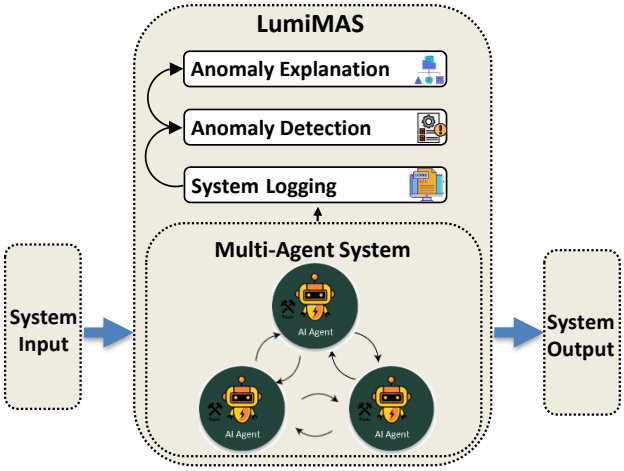

Figure 1: High-level architecture of the LumiMAS observability framework incorporated in a given MAS

selected from a predefined set of event types E. The life cycle of a MAS application is bounded by the events `Application-Started` and `Application-Ended`. Within these boundaries, agent-specific execution spans are recorded via `Agent-Started`(a) and `Agent-Finished`(a), where $a \in A$ denotes an agent identity from the set of all agents A. Throughout the agent's execution, we log every LLM invocation and tool usage as discrete events (`LLM-Call` and `Tool-Usage`) for every iteration, capturing the agent's behavior. The telemetry data collected for each of these events includes operational features detailing agents' behavior (e.g., number of tool invocations), resource consumption (e.g., token usage counts), and semantic information derived from agents' LLM interactions. Each event's monitored features are provided in the supplementary material. Our logging approach is designed to be agnostic to the specific underlying MAS platform (e.g., CrewAI [6], LangGraph [20]).

4.2 Anomaly Detection Layer

The next component of our framework is an anomaly detection layer designed to be lightweight (to facilitate real-time detection) and identify various failures in MAS applications. Our detection method combines engineered features that capture structural and statistical properties of agent behavior and semantics features extracted from the text generated during agent interactions (e.g., LLM outputs). As our method designed to handle long-term dependencies, inter-agent communication patterns, and variable-length logs in a dynamic multi-agent environment. To maintain low latency under these conditions, we employ an LSTM-based autoencoder (AE) architecture, which has proven particularly effective for the anomaly detection task [21, 33]. Under the assumption that failures within MAS exhibit abnormal behavior in agent execution and semantic communication, we propose three AE-based detection approaches: (1) Execution performance indicator (EPI) detection, which utilizes features extracted from each agent's execution, such as execution duration and token count (features are elaborated in

the supplementary material). (2) Semantic detection, which utilizes encoded representations of outputs from LLM interactions (e.g., the agent's reasoning process). (3) Combined latent-space detection, which employs a linear AE trained on the concatenated latent representations from the first two approaches. Figure 2 presents our anomaly detection architecture overview. The AE models are trained to reconstruct features from logs collected using our system logging layer. At inference, the features extracted from the execution logs are propagated through the model, which returns reconstructed representations of the features. The reconstruction error between the model's input and output is computed; a high reconstruction error indicates potential anomalies. The sequential nature of our anomaly detection approach is used to capture the whole system's normal patterns and flow, which helps detect deviations from those patterns. By combining these different representations, our method can detect a wide range of anomalies, without being limited to a specific application or failure type.

4.2.1 Feature Extraction. Given a log, a feature extraction process is being performed. In the EPI detection approach, we extract EPI features such as the number of tools used, execution duration (detailed in the supplementary material) denoted as $X^{<\mathcal{E}>} = (x_1^{<\mathcal{E}>}, ..., x_n^{<\mathcal{E}>})$, where $x^{<\mathcal{E}>}$ is the feature vector of a single agent execution, and n is the number of agent executions.

As MASs generate a large amount of textual data, our semantic detection approach leverages the textual content of LLM's interactions, aiming to capture anomalies in the agent's reasoning process. Each agent's LLM interaction (i.e., the LLM's output) is encoded into a compact vector representation using a sentence-transformer [30]. This embedding sequence is denoted by $X^{<\mathcal{S}>} = (x_1^{<\mathcal{S}>}, ..., x_m^{<\mathcal{S}>})$, where $x^{<\mathcal{S}>}$ is the encoded LLM interaction, and m is the number of LLM interactions.

4.2.2 Model Training. The training of both the EPI ($\mathcal{E}$) and semantic ($\mathcal{S}$) AEs is performed in a similar process. Let $\mathcal{I} \in \{\mathcal{E}, \mathcal{S}\}$ denote the type of AE. In both cases, the features $X^{<\mathcal{I}>}$ are first encoded into a latent representation using LSTM layers ($f^{\mathcal{I}}$), denoted by $\mathcal{Z}^{\mathcal{I}} = f^{\mathcal{I}}(X^{<\mathcal{I}>})$. Then the encoded features are decoded by the decoder ($g^{\mathcal{I}}$) to reconstruct the original input $X^{<\mathcal{I}>\prime} = g^{\mathcal{I}}(\mathcal{Z}^{<\mathcal{I}>})$. The training objective is to minimize the reconstruction error between the model's input and output; thus, the loss function of both approaches can be defined as:

$$\ell_{\mathcal{I}} = d\left(X^{\langle\mathcal{I}\rangle\prime}, X^{\langle\mathcal{I}\rangle}\right) \tag{1}$$

where d is a distance function.

4.2.3 Combined Latent Space AE. Using the latent representations learned by the models, we train a unified AE that integrates both of the previously described approaches. The integration of the two contributes to a more effective and comprehensive representation of system behavior. First, we extract both $X^{<\mathcal{E}>}$ and $X^{<\mathcal{S}>}$ from the execution log. These features are fed into the EPI and semantic encoder's layers, resulting in $\mathcal{Z}^{\mathcal{E}}$ and $\mathcal{Z}^{\mathcal{S}}$, respectively. Once the latent space representations are extracted, they are concatenated into a single feature vector $X^{<C>}$. The new features are encoded into a latent space representation $\mathcal{Z}^{C} = f^{C}(X^{<C>})$ and then decoded back to reconstruct the original input $X^{<C>\prime} = g^{C}(\mathcal{Z}^{C})$, where f^{C} and g^{C} are the model's linear encoder and decoder layers, respectively. Similar to Section 4.2.2, the training objective is to minimize the reconstruction error between the model input and its output:

$$\ell_{C} = d\left(X^{\langle C\rangle\prime}, X^{\langle C\rangle}\right) \tag{2}$$

4.2.4 Failure-Detector Mapping. Each LumiMAS's detector aligns with different signal types and corresponding failure categories within the MAST taxonomy [2]. The Semantic-AE operates over embeddings of agent reasoning, message exchanges, and memory traces, enabling it to capture linguistic and logical inconsistencies, reasoning drift, and other deviations from prompt specifications or agent communication. These correspond to MAST's Specification Issues and Inter-Agent Misalignment failures, such as Disobey Task Specification, Disobey Role Specification, Loss of Conversation History, Task Derailment, Conversation Reset, and Reasoning–Action Mismatch. The EPI-AE, in contrast, consumes execution and performance indicators, including latency, message frequency, and runtime metrics, and is thus sensitive to abnormal execution patterns, resource exhaustion, and unexpected termination. These signals are characteristic of MAST's failures, including Premature Termination, Conversation Reset, and Step Repetition. Finally, the combined detector integrates both semantic and EPI features to uncover inconsistencies between an agent's communicative intent and its observable behavior. This fusion provides greater robustness across the anomaly types. Nevertheless, certain failure modes still benefit more from specialized components when the anomaly is strongly semantic or operation-oriented. Together, these detectors provide complementary coverage of the major MAST failure categories. While LumiMAS achieves broad coverage across failure types, its current architecture is not primarily focused on Task Verification failures (categories 2.2, 3.2, and 3.3 in MAST), which typically require explicit reasoning about task completeness or output validation. As a result, failures such as Incomplete Verification or Incorrect Verification may be indirectly detected through performance anomalies (e.g., token count or latency changes) but are not explicitly modeled.

4.3 Anomaly Explanation Layer

While producing real-time alerts for incoming threats is the primary goal of our framework, providing additional contextual information alongside each detection can help to effectively address the anomaly. Aiming to provide additional context to the alerts rising from the anomaly detection layer LumiMAS features an anomaly explanation layer which incorporate two specialized LMA agents: (1) classification agent and (2) root cause analysis (RCA) agent.

The classification agent analyzes the suspected log, aiming to assign the appropriate category out of a predefined set of vulnerability types (e.g., bias, DPI). Moreover, by incorporating the 'benign' category, the agent can filter cases that do not indicate anomalous behavior, which can reduce false positives. The classification agent's prompt, which is provided in the supplementary material, is composed of four structured elements: (1) the agent's role, (2) the agent's task description that outlines expected behavior, (3) a list of the defined vulnerabilities, and (4) the format of expected response.

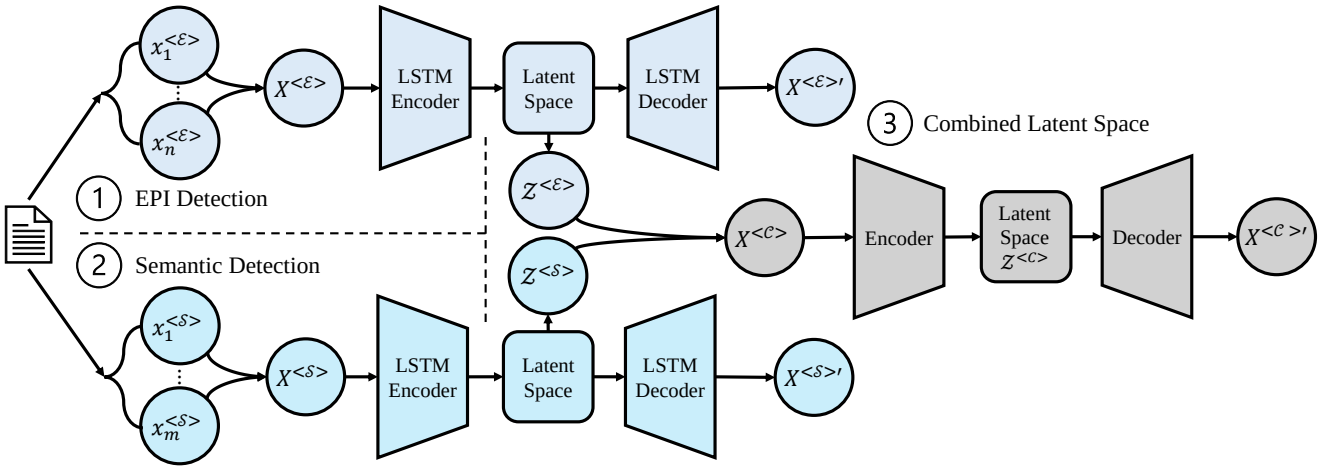

Figure 2: Overview of the anomaly detection architecture showing (1) the EPI-based autoencoder, (2) the semantic-based autoencoder, and (3) a combined detector that integrates both approaches

The RCA agent leverages the classification output to trace the origin of the detected anomaly. Guided by a similarly structured prompt, the RCA agent examines the chronological execution flow and inter-agent interactions to identify the first agent responsible for introducing the anomalous behavior. Its output includes both the root cause agent identifier and a detailed explanation of the causal chain leading to the failure. Working collaboratively, the classification and RCA agents form a cohesive diagnostic pipeline: the first characterizes the nature of the anomaly, and the second locates its origin within the system's execution flow, enabling comprehensive and interpretable MAS failure analysis.

5 EVALUATION

5.1 Evaluation Settings

This subsection outlines the primary settings of our evaluation. Additional details, such as extended hyperparameter configurations and information about the applications used, are provided in the supplementary material.

5.1.1 MAS Applications. We employed seven different MAS applications, each demonstrating a distinct vulnerability, and two MAS platforms, CrewAI [6] and LangGraph [20]. On the CrewAI framework, we employed the Trip Planner and Instagram Post applications [5] to demonstrate the detection of DPI and IPI attacks, respectively. For MP attack detection, we utilized the Real Estate Team application [14], which includes an RAG database. On the LangGraph framework, we employed the GenFic application and our adapted version of the Trip Planner application from [5], focusing on DPI attack detection.

While some vulnerabilities are more straightforward to generate and evaluate (e.g., DPI and IPI), vulnerabilities that originate from inherent limitations of the LLM (e.g., hallucination and bias) are more difficult to trigger. Therefore, we developed FTAs to enable the assessment of detection and mitigation strategies. HalluCheck, our hallucination-tailored application, utilizes a question-answering dataset [29] with three specially designed agents to illustrate the effect on the features (e.g., increased run time) in a MAS application when a hallucination occurs. Similarly, BiasCheck enables illustrating the effect on the features in a MAS when bias occurs, leveraging a stereotype questions dataset [27].

5.1.2 Failure Generation. While the failures in our FTAs result from the application's design and datasets used, the faults in the other applications are induced in different ways, aligned with the type of vulnerability. The generation of DPI, IPI, and MP attacks was inspired by [7, 37] and allowed us to generate realistic attack scenarios. In the first DPI scenario, we crafted a prompt that forces the agent's LLM to produce outputs in an invalid format. As a result, the agent fails to parse the response, which leads to increased computational load, excessive token usage, and longer execution time. In the second DPI, we concatenated misinformation with the user input, causing the agent to base its reasoning and outputs on false information aligned with the attacker's intent. Another DPI variant acted as a form of backdoor attack, injecting into the prompt condition tied to the agent's identity (e.g., name). Once triggered, the agent's behavior changes to serve the attacker's intent and affects the system's overall output. To demonstrate the IPI scenario, we set up a malicious web server containing HTML content with false instructions. This attack aimed to lure the agent into repeatedly following deceptive URLs, resulting in another form of system exhaustion. Lastly, in the MP scenario, we poisoned the RAG database that the system's agents rely on, which results in poisoned information being retrieved.

5.1.3 Data Collection and Splitting. In our monitoring and logging layer, over 2,000 benign scenarios in each examined application were simulated, resulting in sets of benign logs. The test and validation sets each consisted of 200 logs, with half anomalous (containing failures) and the other half benign. The sets were crafted to ensure that no overlaps occur between the defined sets.

5.1.4 LLM Configuration. To demonstrate the robustness of our framework, we evaluate its effectiveness using two different LLMs as the agents' underlying models: OpenAI GPT-4o mini [18] and

OpenAI o3-mini [25]. GPT-4o mini is a lightweight version of GPT-4o, optimized for speed and efficiency, with strong general performance. o3-mini is a compact LLM that has been shown to perform efficiently across various natural language processing tasks.

5.1.5 Baselines. As no prior work had been done to detect failures in MASs, we utilized the LLM-as-a-judge paradigm [39] with an engineered prompt for the anomaly detection task. Inspired by Zhuge et al. [41], we also implemented the agent-as-a-judge variant, with the task and agent definition based on the same engineered prompt. We equip the agent with scraping and searching tools, enabling it to retrieve additional information to improve its effectiveness in detecting failures. GPT-4o [18] was used as the backbone of our baselines, analyzing each agent's input and output. Consequently, the prediction of the baseline approaches also indicates which agent is responsible for the failure, and this information is used in our RCA evaluation. Additional results using the smaller and more cost-efficient GPT-4o-mini [18] are provided in the supplementary material. As a non-LLM baseline, we implemented LogBERT [12] for log-based anomaly detection, providing a purely sequential modeling reference against our anomaly detectors, more details are provided in the supplementary material. Furthermore, we evaluate existing hallucination and bias detection strategies for our FTA apps. For hallucination detection, we adopted the consistency comparison approach [22], in which each agent was instructed to repeat the task 3 or 5 additional times, and the original result was considered consistent (i.e., benign) if it aligned with the majority of the outputs. Regarding bias, we employed toxic-bert [15] to assess the bias in each corresponding agent.

5.1.6 Metrics. To evaluate failure detection performance, we used several standard metrics applied in binary classification tasks (accuracy, precision, recall, F1 score, and false positive rate (FPR)), as well as metrics measuring the solutions' overhead resource consumption (average inference time and token count for models that require payment). For the RCA evaluation, we used the root cause accuracy (RA) metric from [38, 40]. In the results tables, the best performance in each column is marked in bold, and the second-best is underlined. The arrows (↑, ↓) indicate whether the optimal value is higher or lower.

5.1.7 Implementation Details. We implemented our method using Python 3.11 and PyTorch 2.6, and the models were trained on an RTX 4090 GPU. Each application had its own set of hyperparameters, and all models were trained using the Adam optimizer. For the semantic features, the textual features (i.e., the agent's LLM interactions) were embedded using the all-MiniLM-L6-v2 model based on sentence-transformers [30]. A bidirectional LSTM architecture was used in the AE model. The anomaly detection classification threshold was selected based on the F1 score on the validation set, while the models' other hyperparameters were tuned to minimize the reconstruction error on the benign validation set. The distance function used in all three models' loss function was the mean square error. The exact hyperparameters are provided in the supplementary material.

5.2 Evaluation Results

This subsection outlines our evaluation results of the anomaly detection method, RCA, and classification agents. The presented results were obtained using the CrewAI apps, with GPT-4o-mini as the underlying model, while additional results obtained using o3-mini and on the LangGraph framework are provided in the supplementary material.

5.2.1 Anomaly Detection Results. Table 1 presents the anomaly detection results of the five types of vulnerabilities evaluated (rows), along with each detection approach's performance and overhead. Note that the average performance across the three different DPI attacks is presented for the DPI vulnerability. The results were obtained by averaging five independent runs using different seeds, and are reported as the mean and standard deviation. Our combined approach demonstrates strong performance across multiple applications, achieving the lowest average FPR (0.267) and highest average accuracy (0.750), precision (0.752), and F1 (0.754) among the examined baselines while maintaining competitive average recall (0.768). Moreover, it is one of the most efficient approach in terms of runtime, with an average decision time of 0.057 seconds; in contrast, the average inference time for the fastest LLM-based approaches is 8.448 seconds, which is 148 times slower. The efficiency of our method remains consistent even in larger (with more agents) scenarios, with only a negligible increase in runtime, as demonstrated in the supplementary material. Although the LogBERT approach exhibits lower runtime compared to LumiMAS's combined approach, it achieves a substantially lower performance with an average accuracy of 0.632 and F1 of 0.668 across all failure types. LogBERT performs better in cases where anomalies are reflected in the log structure (e.g., in the IPI scenario), as it leverages sequential event patterns to detect structural deviations within the log stream. However, it performs poorly on failures that are not reflected in the log structure, arising instead from semantic shifts or performance-related symptoms at the system level (e.g., DPI, hallucination and bias). Furthermore, our anomaly detection approach imposes no inference cost regarding token usage, unlike the LLM-based baselines that rely on external API calls. The combined variant achieves the highest F1 score in detecting hallucinations (0.733) and maintains consistently competitive performance across almost all failure types. However, this approach is relatively less effective at detecting bias, possibly because such bias stems from the model's training data, causing biased and unbiased logs to exhibit similar characteristics. For the MP attack scenario, the semantic approach and the LLM-based examined baselines perform better than the EPI approach due to the textual nature of the attack. Although in some cases, the EPI and semantic approaches perform better than the combined approach, the results indicate that the combined approach was the most stable. This suggests that integrating low-level system features and high-level textual LLM interactions enhances the robustness of anomaly detection in MASs.

5.2.2 Root Cause Analysis Results. The RCA results are presented in Figure 3, which compares the performance of the proposed RCA agent across the five vulnerability types with the examined baselines. The results indicate that our RCA agent demonstrates competitive or superior performance across the evaluated vulnerabilities.

Table 1: Anomaly detection results on CrewAI applications using GPT-4o-mini as the underlying model

Vulnerability	Method	Performance					Overhead	
		Accuracy ↑	F1 ↑	Recall ↑	Precision ↑	FPR ↓	Latency ↓	Token Count ↓
Hallucination	Consistency of 3	0.681 ± 0.020	0.721 ± 0.015	0.826 ± 0.018	0.641 ± 0.018	0.464 ± 0.033	12.941	5312.4
	Consistency of 5	0.684 ± 0.009	0.727 ± 0.008	**0.842 ± 0.013**	0.640 ± 0.006	0.474 ± 0.009	24.072	8848.4
	LLM-as-a-judge	0.574 ± 0.010	0.332 ± 0.025	0.212 ± 0.019	0.767 ± 0.022	**0.064 ± 0.005**	6.215	2414.0
	Agent-as-a-judge	0.586 ± 0.033	0.406 ± 0.059	0.284 ± 0.051	0.717 ± 0.073	0.112 ± 0.030	9.617	1436329.0
	LogBERT	0.506 ± 0.005	0.320 ± 0.312	0.422 ± 0.509	0.575 ± 0.068	0.410 ± 0.520	**0.006**	-
	EPI (Ours)	0.755 ± 0.015	0.722 ± 0.013	0.638 ± 0.040	0.838 ± 0.057	0.128 ± 0.059	0.007	-
	Semantic (Ours)	0.716 ± 0.009	0.689 ± 0.016	0.632 ± 0.048	0.763 ± 0.034	0.200 ± 0.054	0.013	-
	Combined (Ours)	**0.765 ± 0.015**	**0.733 ± 0.016**	0.646 ± 0.030	**0.850 ± 0.039**	0.116 ± 0.042	0.016	-
Bias	toxic-bert	0.550 ± 0.000	0.683 ± 0.000	**0.970 ± 0.000**	0.527 ± 0.000	0.870 ± 0.000	0.251	
	LLM-as-a-judge	**0.907 ± 0.008**	**0.909 ± 0.008**	0.926 ± 0.011	0.892 ± 0.010	0.112 ± 0.011	5.297	2229.0
	Agent-as-a-judge	0.894 ± 0.015	0.892 ± 0.015	0.876 ± 0.017	**0.909 ± 0.023**	**0.088 ± 0.024**	10.129	1253684.0
	LogBERT	0.508 ± 0.027	0.640 ± 0.034	0.886 ± 0.139	0.506 ± 0.019	0.870 ± 0.169	0.008	-
	EPI (Ours)	0.586 ± 0.010	0.605 ± 0.012	0.634 ± 0.017	0.578 ± 0.008	0.462 ± 0.004	**0.005**	-
	Semantic (Ours)	0.612 ± 0.009	0.632 ± 0.013	0.666 ± 0.023	0.601 ± 0.007	0.442 ± 0.015	0.020	-
	Combined (Ours)	0.664 ± 0.027	0.661 ± 0.020	0.654 ± 0.015	0.668 ± 0.033	0.326 ± 0.048	0.023	-
DPI	LLM-as-a-judge	0.705 ± 0.016	0.750 ± 0.013	**0.903 ± 0.015**	0.644 ± 0.014	0.492 ± 0.025	7.268	4413.3
	Agent-as-a-judge	0.729 ± 0.023	0.760 ± 0.021	0.885 ± 0.021	0.670 ± 0.022	0.427 ± 0.036	18.275	3682504.8
	LogBERT	0.572 ± 0.072	0.665 ± 0.059	0.868 ± 0.119	0.550 ± 0.052	0.724 ± 0.251	**0.005**	-
	EPI (Ours)	0.713 ± 0.018	0.734 ± 0.013	0.826 ± 0.011	0.666 ± 0.021	0.400 ± 0.038	**0.005**	-
	Semantic (Ours)	**0.815 ± 0.010**	**0.819 ± 0.011**	0.865 ± 0.022	**0.784 ± 0.013**	**0.234 ± 0.023**	0.071	-
	Combined (Ours)	0.802 ± 0.021	0.803 ± 0.026	0.842 ± 0.038	0.775 ± 0.021	0.238 ± 0.028	0.076	-
IPI	LLM-as-a-judge	0.547 ± 0.004	0.681 ± 0.002	0.966 ± 0.009	0.526 ± 0.003	0.872 ± 0.015	17.987	6642.0
	Agent-as-a-judge	0.614 ± 0.020	0.698 ± 0.013	0.890 ± 0.012	0.574 ± 0.015	0.662 ± 0.036	34.618	9895764.8
	LogBERT	**0.959 ± 0.022**	**0.960 ± 0.021**	0.970 ± 0.017	**0.951 ± 0.044**	**0.052 ± 0.049**	**0.007**	-
	EPI (Ours)	0.943 ± 0.011	0.944 ± 0.012	**0.972 ± 0.029**	0.919 ± 0.012	0.086 ± 0.015	0.010	-
	Semantic (Ours)	0.910 ± 0.013	0.910 ± 0.012	0.912 ± 0.008	0.909 ± 0.023	0.092 ± 0.026	0.079	-
	Combined (Ours)	0.941 ± 0.033	0.941 ± 0.033	0.942 ± 0.039	0.941 ± 0.040	0.060 ± 0.041	0.102	-
MP	LLM-as-a-judge	**0.797 ± 0.012**	**0.814 ± 0.011**	**0.890 ± 0.014**	**0.750 ± 0.011**	0.296 ± 0.015	7.835	3701.0
	Agent-as-a-judge	0.746 ± 0.013	0.749 ± 0.016	0.758 ± 0.028	0.740 ± 0.011	**0.266 ± 0.015**	10.484	1770939.8
	LogBERT	0.734 ± 0.009	0.765 ± 0.001	0.864 ± 0.036	0.687 ± 0.023	0.396 ± 0.054	0.004	-
	EPI (Ours)	0.523 ± 0.011	0.626 ± 0.019	0.800 ± 0.047	0.515 ± 0.006	0.754 ± 0.030	**0.003**	-
	Semantic (Ours)	0.715 ± 0.019	0.714 ± 0.019	0.712 ± 0.048	0.719 ± 0.036	0.282 ± 0.065	0.027	-
	Combined (Ours)	0.477 ± 0.068	0.536 ± 0.065	0.608 ± 0.097	0.483 ± 0.058	0.654 ± 0.114	0.032	-

Notably, it outperforms the baselines in cases involving adversarial attacks, highlighting its effectiveness in tracing failures that stem from external manipulations. The RCA agent exhibits slightly poorer performance on vulnerabilities arising from the inherent limitations of the agent's LLM (i.e., bias and hallucination), while the baselines retain a marginal advantage. These results underscore the RCA agent's particular strength in scenarios requiring reasoning over inter-agent dynamics and adversarial behaviors, while suggesting that further refinement may be needed to better capture subtler internal model failures.

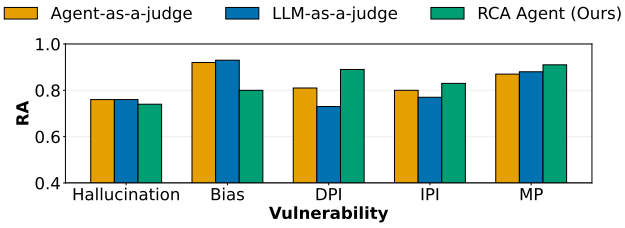

Figure 3: Root cause analysis (RCA) results obtained using CrewAI apps with GPT-4o-mini as the underlying model

5.2.3 Anomaly Classification Results. The classification agent evaluation results, as well as their impact on the final anomaly detection decision (which are elaborated in the supplementary material), indicate that the classifier effectively filters false positives, resulting in an approximately 60% reduction in the FPR on average. It also achieved high accuracy in identifying biased cases with a score of 78%. Nonetheless, the agent tends to classify instances of adversarial attacks as hallucinations inaccurately. These misclassifications are likely caused by attacks that undermine the alignment of the agent's output with its input, resulting in behavior similar to hallucinations. A deeper examination of misclassifications patterns revealed several notable trends. In particular, and consistent with the MAST taxonomy [2], cases where the classification agent mislabeled adversarial inputs as hallucination reflect a Task Derailment

failure mode. This phenomenon likely arises because adversarial behaviors often mimic benign reasoning errors, producing patterns that appear semantically coherent yet subtly deviate from the intended task. As a result, the classifier interprets these deliberate manipulations as ordinary hallucinations rather than malicious, highlighting the challenge of distinguishing deceptive intent from naturally occurring drift within multi-agent interactions.

5.2.4 Anomaly Explanation Evaluation. We evaluated the explanations generated by the anomaly explanation layer using automated rule-based scoring metric and human validation across the seven examined failures. The explanations were assessed on three dimensions: (1) identifying the underlying failure patterns, (2) providing relevant justifications, and (3) offering complete accounts of both the classification and root cause. The results of the automated assessment show that the explanations achieved an average score of 7.5 out of 10, indicating informative and relevant explanations. Moreover, we performed an application-grounded human evaluation following the framework of Doshi-Velez and Kim [9]. Team of domain experts independently rated a subset of explanations using the same rubrics of failure-pattern detection, evidence quality, and reasoning completeness. Each expert received the log snippet, the system-generated explanation, and the ground-truth label for every case. Mean expert scores were computed for each dimension and failure type. The human review confirmed that the explanations aligned with the expected failure characteristics and effectively supported the interpretation of the detected anomalies. Full details are provided in the supplementary material.

6 DISCUSSION

6.1 LumiMAS Limitations

While LumiMAS demonstrated strong anomaly detection performance, it has some limitations. One of the primary drawbacks of the framework is its reliance on training the anomaly detection model before deployment, and it may require retraining for application updates to incorporate new behaviors. However, the training process is lightweight, and once trained, it has low inference time, making it well-suited for real-time anomaly detection. Training can be scheduled periodically with minimal overhead, and certain types of updates can be handled without retraining (as detailed in the supplementary material). Moreover, the classification agent's predefined vulnerability types limit its flexibility but enable precise categorization. The supplementary material evaluates a variant without predefined types, highlighting this trade-off.

6.2 Resource Consumption

To address failures as they occur, real-time anomaly detection is essential. As demonstrated in Section 5, LumiMAS can detect failures in real-time (under 0.06 seconds), making it suitable for real-time detection and large-scale applications. The training process is also efficient: while GPU acceleration reduces training time to 0.076 GPU-hours, the model can be trained entirely on a CPU in just 1.133 CPU-hours with minimal resource consumption. Supporting experiments are provided in the supplementary material.

6.3 Retraining and Data Drift

LumiMAS's anomaly detection is trained on normal MAS behavior, and its transferability depends on behavioral alignment across applications. Using it for an unseen application will be possible if the two applications are strongly aligned. As shown in the supplementary material, some changes, such as tool replacements, are more critical and necessitate retraining, whereas others, like task modifications with similar semantic meaning, have minimal impact and typically do not require retraining. The retraining frequency and data needs depend on the application complexity and the scale of the change to the application. As with all training-based ML models, distribution drift may occur and is addressed by periodic retraining. Importantly, as demonstrated in the supplementary material, the training procedure is lightweight, enabling efficient and regular retraining.

6.4 Agnostic Framework

As MAS platforms continue to evolve, the need for observability in such systems becomes increasingly critical to safely integrating AI agents into production applications. LumiMAS was designed to be platform-agnostic as the system logging layer collects only generic MAS event and data types, such as LLM call information, timestamps, execution metadata, and token-level statistics. These signals are common across most modern LLM-based MAS frameworks and do not rely on framework-specific internal logic. We tested LumiMAS on two different MAS platforms (CrewAI [6], LangGraph [20]), which differ in execution flow and operating style. This provides an indication that our solution is applicable for diverse MASs.

6.5 EPI vs. Semantic Detection

While the combined anomaly detection approach generally achieved the most balanced results, there are scenarios in which one of its constituent approaches achieved superior results due to the nature of the failure. The semantic approach performed better in cases where the failure is primarily textual (e.g., MP). In contrast, the EPI approach performed better in cases where the failure impacts system-level features such as runtime (e.g., exhaustion attacks). Moreover, we evaluate an alternative fusion strategy for combining EPI and semantic features. This additional experiment (described in detail in the supplementary material) further clarifies how different fusion choices affect performance across failure types. This underscores the importance of selecting the appropriate anomaly detection approach based on specific user preferences.

7 CONCLUSION

In this paper, we introduce LumiMAS, a novel and efficient platform-agnostic framework for observability and failure detection in MASs. The extensive evaluation demonstrates LumiMAS's efficiency, characterized by a low resource consumption and real-time anomaly detection capabilities while maintaining low FPR and high accuracy. We also introduce novel FTAs, enabling a reliable benchmark for failures such as hallucinations and bias in MASs. Moreover, by publishing our dataset, our research enables other studies to further enhance the capabilities presented in this paper. Future work may focus on more advanced monitoring and failure detection techniques in MASs as they evolve.

REFERENCES

[1] Roy Betser, Shamik Bose, Amit Giloni, Chiara Picardi, Sindhu Padakandla, and Roman Vainshtein. 2026. AgenTRIM: Tool Risk Mitigation for Agentic AI. *arXiv preprint arXiv:2601.12449* (2026).

[2] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657* (2025).

[3] Alan Chan, Carson Ezell, Max Kaufmann, Kevin Wei, Lewis Hammond, Herbie Bradley, Emma Bluemke, Nitarshan Rajkumar, David Krueger, Noam Kolt, et al. 2024. Visibility into AI agents. In *Proceedings of the 2024 ACM conference on fairness, accountability, and transparency*. 958–973.

[4] Chi-Min Chan, Jianxuan Yu, Weize Chen, Chunyang Jiang, Xinyu Liu, Weijie Shi, Zhiyuan Liu, Wei Xue, and Yike Guo. 2024. Agentmonitor: A plug-and-play framework for predictive and secure multi-agent systems. *arXiv preprint arXiv:2408.14972* (2024).

[5] Crew AI Inc. 2024. CrewAI Examples. https://github.com/crewAIInc/crewAI-examples.

[6] CrewAI Inc. 2025. CrewAI Documentation. https://docs.crewai.com/.

[7] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. 2025. Ai agents under threat: A survey of key security challenges and future pathways. *Comput. Surveys* 57, 7 (2025), 1–36.

[8] Liming Dong, Qinghua Lu, and Liming Zhu. 2024. AgentOps: Enabling Observability of LLM Agents. *arXiv preprint arXiv:2411.05285* (2024).

[9] Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* (2017).

[10] Haishuo Fang, Xiaodan Zhu, and Iryna Gurevych. [n.d.]. Inferact: Inferring safe actions for llm-based agents through preemptive evaluation and human feedback, 2024. *URL https://arxiv. org/abs/2407.11843* ([n. d.]).

[11] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*. 79–90.

[12] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.

[13] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680* (2024).

[14] Brandon Hancock. 2024. CrewAI RAG Deep Dive. https://github.com/bhancockio/crewai-rag-deep-dive.

[15] Laura Hanu and Unitary team. 2020. Detoxify. Github. https://github.com/unitaryai/detoxify.

[16] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2023. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*.

[17] Wenyue Hua, Xianjun Yang, Mingyu Jin, Zelong Li, Wei Cheng, Ruixiang Tang, and Yongfeng Zhang. 2024. Trustagent: Towards safe and trustworthy llm-based agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 10000–10016.

[18] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).

[19] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM computing surveys* 55, 12 (2023), 1–38.

[20] LangGraph. 2023. LangGraph: Building stateful multi-actor applications with LLMs. https://github.com/langchain-ai/langgraph.

[21] Younjeong Lee, Chanho Park, Namji Kim, Jisu Ahn, and Jongpil Jeong. 2024. LSTM-autoencoder based anomaly detection using vibration data of wind turbines. *Sensors* 24, 9 (2024), 2833.

[22] Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*. 9004–9017.

[23] Silen Naihin, David Atkinson, Marc Green, Merwane Hamadi, Craig Swift, Douglas Schonholtz, Adam Tauman Kalai, and David Bau. 2023. Testing language model agents safely in the wild. *arXiv preprint arXiv:2311.10538* (2023).

[24] Roberto Navigli, Simone Conia, and Björn Ross. 2023. Biases in large language models: origins, inventory, and discussion. *ACM Journal of Data and Information Quality* 15, 2 (2023), 1–21.

[25] OpenAI. 2025. OpenAI o3-mini System Card. https://cdn.openai.com/o3-mini-system-card-feb10.pdf. System card technical report.

[26] OWASP Foundation. 2025. OWASP Top 10 for Large Language Model Applications. https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/.

[27] Alicia Parrish, Angelica Chen, Nikita Nangia, Vishakh Padmakumar, Jason Phang, Jana Thompson, Phu Mon Htut, and Samuel Bowman. 2022. BBQ: A hand-built bias benchmark for question answering. In *Findings of the Association for Computational Linguistics: ACL 2022*. 2086–2105.

[28] Pierre Peigné, Mikolaj Kniejski, Filip Sondej, Matthieu David, Jason Hoelscher-Obermaier, Christian Schroeder de Witt, and Esben Kran. 2025. Multi-agent security tax: Trading off security and collaboration capabilities in multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 27573–27581.

[29] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[30] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 3982–3992.

[31] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. 2023. Identifying the risks of lm agents with an lm-emulated sandbox. *arXiv preprint arXiv:2309.15817* (2023).

[32] Madhulika Srikumar, Kasia Chmielinski, Jacob Pratt, Carolyn Ashurst, Chloé Bakalar, William Bartholomew, Rishi Bommasani, Peter Cihon, Rebecca Crootof, Mia Hoffmann, Ruchika Joshi, Maarten Sap, and Caleb Withers. 2025. Prioritizing Real-Time Failure Detection in AI Agents. https://partnershiponai.org/resource/prioritizing-real-time-failure-detection-in-ai-agents/.

[33] Yuanyuan Wei, Julian Jang-Jaccard, Wen Xu, Fariza Sabrina, Seyit Camtepe, and Mikael Boulic. 2023. LSTM-autoencoder-based anomaly detection for indoor air quality time-series data. *IEEE Sensors Journal* 23, 4 (2023), 3787–3800.

[34] Zhenjie Xu, Wenqing Chen, Yi Tang, Xuanying Li, Cheng Hu, Zhixuan Chu, Kui Ren, Zibin Zheng, and Zhichao Lu. 2025. Mitigating social bias in large language models: A multi-objective approach within a multi-agent framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 25579–25587.

[35] Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, et al. 2024. R-judge: Benchmarking safety risk awareness for llm agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 1467–1490.

[36] Cong Zhang, Xin Deik Goh, Dexun Li, Hao Zhang, and Yong Liu. 2025. Planning with multi-constraints via collaborative language agents. In *Proceedings of the 31st International Conference on Computational Linguistics*. 10054–10082.

[37] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. 2024. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. *arXiv preprint arXiv:2410.02644* (2024).

[38] Wei Zhang, Hongcheng Guo, Jian Yang, Zhoujin Tian, Yi Zhang, Yan Chaoran, Zhoujun Li, Tongliang Li, Xu Shi, Liangfan Zheng, et al. 2024. mABC: multi-Agent Blockchain-Inspired Collaboration for root cause analysis in micro-services architecture. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 4017–4033.

[39] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.

[40] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2023. D-bot: Database diagnosis system using large language models. *arXiv preprint arXiv:2312.01454* (2023).

[41] Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. 2024. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934* (2024).